Implementing GeoCordinateWatcher as a Reactive Service

This article explains how to use the Windows Phone Reactive Extensions (Rx) library to subscribe to geo location events.

Introduction

The Reactive Extensions (Rx) is a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query operators. Using Rx, developers ***represent*** asynchronous data streams with Observables, ***query*** asynchronous data streams using LINQ operators, and ***parameterize*** the concurrency in the asynchronous data streams using Schedulers. Simply put, Rx = Observables + LINQ + Schedulers." – from the MSDN page.

The library also provides a considerable number of helper methods that make it easy to wrap events into observables - events become first class citizens that can be passed around and composed as needed in a very simple way.

Wrapping GeoCoordinateWatcher as a Reactive Service

Wrapping the GeoCoordinateWatcher as a reactive service is quite simple. All it takes is creating observables and exposing the events as observables:

```
public class GeoCoordinateReactiveService : IGeoCoordinateReactiveService, IDisposable
{
    private readonly GeoCoordinateWatcher geoCoordinateWatcher = new
GeoCoordinateWatcher();

    public GeoCoordinateReactiveService()
    {
        this.StatusObservable = Observable
            .FromEventPattern<GeoPositionStatusChangedEventArgs>(
                handler => geoCoordinateWatcher.StatusChanged += handler,
                handler => geoCoordinateWatcher.StatusChanged -= handler);

        this.PositionObservable = Observable
            .FromEventPattern<GeoPositionChangedEventArgs<GeoCoordinate>>(
                handler => geoCoordinateWatcher.PositionChanged += handler,
                handler => geoCoordinateWatcher.PositionChanged -= handler);
    }

    public IObservable<EventPattern<GeoPositionStatus> StatusObservable { get; private
set; }

    public IObservable<EventPattern<GeoPosition<GeoCoordinate>> PositionObservable {
get; private set; }
}
```

And now, instead of the StatusChanged and PositionChanged events we have respectively the **StatusObservable** and **PositionObservable** as a stream of EventPattern<TEventArgs> instances.

But the **EventPattern<TEventArgs>** class includes the source of the event and an the event arguments in properties which is far more than we need in this case. With normal LINQ operators we can convert thes streams of **EventPattern<TEventArgs>** instances in streams of the desired values.

```
public class GeoCoordinateReactiveService : IGeoCoordinateReactiveService, IDisposable
{
    private readonly GeoCoordinateWatcher geoCoordinateWatcher = new
GeoCoordinateWatcher();
```


(C) Copyright Nokia 2014. All rights reserved.

```
    public GeoCoordinateReactiveService()
    {
        this.StatusObservable = Observable
            .FromEventPattern<GeoPositionStatusChangedEventArgs>(
                handler => geoCoordinateWatcher.StatusChanged += handler,
                handler => geoCoordinateWatcher.StatusChanged -= handler)
            .Select(ep => ep.EventArgs.Status);

        this.PositionObservable = Observable
            .FromEventPattern<GeoPositionChangedEventArgs<GeoCoordinate>>(
                handler => geoCoordinateWatcher.PositionChanged += handler,
                handler => geoCoordinateWatcher.PositionChanged -= handler)
            .Select(ep => ep.EventArgs.Position);
    }

    public IObservable<GeoPositionStatus> StatusObservable { get; private set; }

    public IObservable<GeoPosition<GeoCoordinate>> PositionObservable { get; private
set; }
}
```

Using The GeoCoordinateReactiveService

Having created an Rx wrapper over the GeoCoordinateWatcher, now we can use it on an application.

This simple application will display the status of the service, the position and the distance traveled.

For the purpose of this application the service will be exposed as a singleton property of the App class:

```
public partial class App : Application
{
    // ...

    public static IGeoCoordinateReactiveService GeoCoordinateService { get; private set;
}

    public App()
    {
        // ...

        InitializePhoneApplication();

        // ...
    }

    // ...

    private void InitializePhoneApplication()
    {
        // ...

        GeoCoordinateService = new GeoCoordinateReactiveService();

        // ...

    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    // ...

}
```

Getting the status of the service is very simple. It just requires subscribing to the **StatusObservable**. Since we want to display the status, we need to observe it on the dispatcher before:

```
App.GeoCoordinateService.StatusObservable
    .ObserveOnDispatcher()
    .Subscribe(this.OnStatusChanged);
```

For the position we do the same with the **PositionObservable**:

```
App.GeoCoordinateService.PositionObservable
    .ObserveOnDispatcher()
    .Subscribe(this.OnPositionChanged);
```

The distance traveled would seem a bit more complicated because we need to keep track of the last position and calculate the distance traveled on every position change. But this is where the **Rx** excels with its query operators. If we combine the position observable with the position observable having skipped one position with the zip operator we end up with an observable with the current and previous position. And if we apply a selector, we get the traveled distance:

```
App.GeoCoordinateService.PositionObservable
    .Zip(
        App.GeoCoordinateService.PositionObservable.Skip(1),
        (p1, p2) => p1.Location.GetDistanceTo(p2.Location))
    .ObserveOnDispatcher()
    .Subscribe(this.OnDistanceChanged);
```

You can find the complete implementation here.

Resources

- The GeoCoordinateWatcher As A Reactive Service Project
- The Reactive Extensions (Rx)... on MSDN
- Rx (Reactive Extensions) on CodePlex
- NuGet Pakages
  - Reactive Extensions - Interfaces Library
  - Reactive Extensions - Core Library
  - Reactive Extensions - Query Library
  - Reactive Extensions - Platform Services Library
  - Reactive Extensions - Main Library
  - Reactive Extensions - XAML Support Library
  - Reactive Extensions - Silverlight Helpers
- Using Rx
- Reactive Extensions (Rx) Forum
- Reactive Extensions Team Blog
- MS Open Tech Open Sources Rx (Reactive Extensions) – a Cure for Asynchronous Data Streams in Cloud Programming

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.